

Hybrid Mobile Applications - ICD0018

TalTech IT College, Andres Käver, 2021-2022, Fall semester

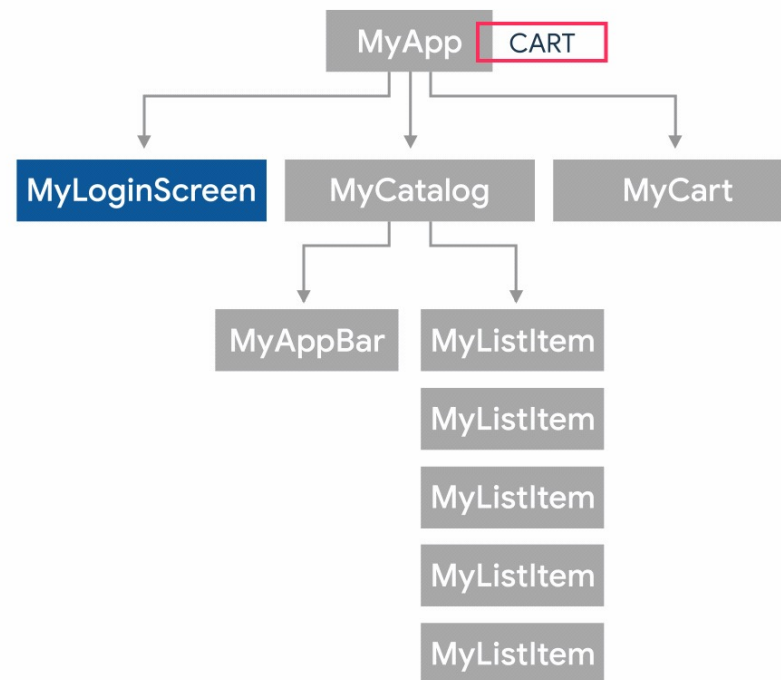
Email: andres.kaver@taltech.ee



Flutter - State

2

A mobile app UI mockup showing a 'Welcome' screen. The screen has a blue header with the word 'Welcome'. Below the header, there is a 'Login' section with two input fields: one for 'Login' and one for 'Password'. At the bottom of the login section is a blue button labeled 'Enter'.



- ▶ Flutter UI is function of state - $UI = f(state)$
- ▶ Declarative style vs imperative
 - ▶ Declarative - Flutter builds its user interface to reflect the current state of your app
- ▶ There is no `TextField.setText()`
- ▶ For example, in Flutter it's okay to rebuild parts of your UI from scratch instead of modifying it. Flutter is fast enough to do that, even on every frame if needed.

Flutter - State

4

- ▶ State - data needed to rebuild UI at any moment in time
- ▶ Ephemeral state - state contained in a single widget
 - ▶ `setState()`
- ▶ App state – state shared across many parts of app (sometimes also called shared state)
 - ▶ Package “provider”
- ▶ Keep the state above the widgets that use it (lifting state up)

Flutter - state

5

- ▶ pubspec.yaml
dependencies:
 flutter:
 sdk: flutter
 provider: ^6.0.0

Flutter - state

6

- ▶ NB! Widgets are immutable – they get destroyed and replaced by new ones.
- ▶ Dart's functions are first class objects, so you can pass callbacks around.

```
@override
Widget build(BuildContext context) {
  return SomeWidget(
    // Construct the widget, passing it a reference to the method below.
    MyListItem(myTapCallback),
  );
}

void myTapCallback(Item item) {
  print('user tapped on $item');
}
```

Flutter - State

7

- ▶ Passing around callbacks – there is lot of them - finally. No good.
- ▶ Flutter has special low-level widgets for state handling
 - ▶ InheritedWidget, InheritedNotifier, InheritedModel
- ▶ Easy to use wrapper around these is
 - ▶ Provider

Flutter - Provider

8

- ▶ `ChangeNotifier` provides change notification to its listeners. You can subscribe to its changes. (It is a form of `Observable`).
- ▶ `ChangeNotifierProvider` is the widget that provides an instance of a `ChangeNotifier` to its descendants.
- ▶ Consumer – get access to state, widget is recreated when state changes
- ▶ `Provider.of` – get access to state, widget is NOT recreated when state changes

Flutter - Model

► POCO (PODO?)

```
class Counter {  
  bool bit0 = false;  
  bool bit1 = false;  
  bool bit2 = false;  
  
  Counter({this.bit2, this.bit1, this.bit0});  
  
  void addToBit0() {  
    bit0 = !bit0;  
    if (!bit0){  
      addToBit1();  
    }  
  }  
  
  void addToBit1(){  
    bit1 = !bit1;  
    if (!bit1){  
      addToBit2();  
    }  
  }  
  
  void addToBit2(){  
    bit2 = !bit2;  
  }  
}
```

Flutter - ChangeNotifier

10

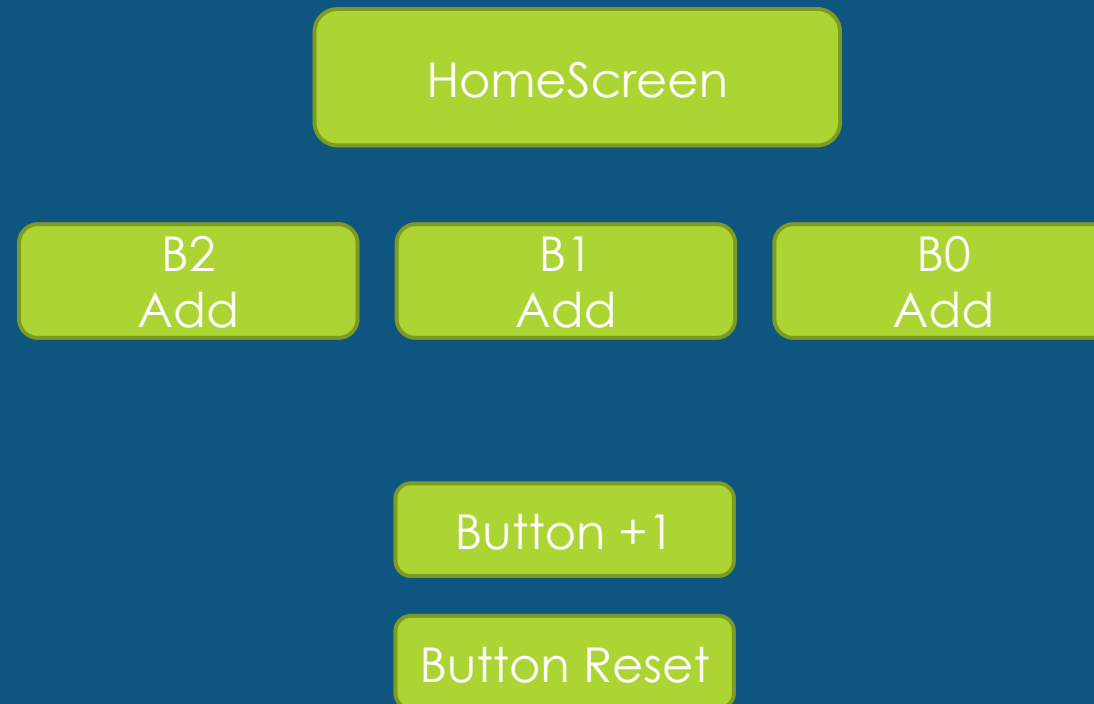
```
import 'package:flutter/material.dart';
import '../models/counter.dart';
class CounterModel extends ChangeNotifier {
  final Counter _counter = Counter();

  bool get bit0 => _counter.bit0;
  bool get bit1 => _counter.bit1;
  bool get bit2 => _counter.bit2;

  void addToBit0() {
    _counter.addToBit0();
    notifyListeners();
  }
  ...
  void add(){
    _counter.addToBit0();
    notifyListeners();
  }
  void reset(){
    _counter.bit0 = false; _counter.bit1 = false; _counter.bit2 = false;
    notifyListeners();
  }
}
```

Flutter – UI – state lifting

11



Flutter - ChangeNotifierProvider

12

► State in HomeScreen

```
Widget build(BuildContext context) {  
  return MaterialApp(  
    title: 'Welcome',  
    home: Scaffold(  
      appBar: AppBar(  
        title: Text('Binary counter'),  
      ),  
      body: ChangeNotifierProvider<CounterModel>(  
        builder: (context) => CounterModel(),  
        child: Center(  

```

Flutter - Consumer

13

- ▶ Widget is recreated on every state change!

```
class SingleBitt extends StatelessWidget {  
  final int bitNo;  
  
  SingleBitt({@required this.bitNo});  
  
  @override  
  Widget build(BuildContext context) {  
    return Consumer<CounterModel>(  
      builder: (context, state, child) {  
        return Container(  
...  
          child: Column(  
            mainAxisAlignment: MainAxisAlignment.min,  
            children: <Widget>[  
              Text(  
                this.getBitStr(state),
```

Flutter – Provider.Of

14

- ▶ Widget is not recreated on state change (possibly)!

```
class ButtonAdd extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return RaisedButton(  
      child: Text('Add 1'),  
      onPressed: () {  
        Provider.of<CounterModel>(context, listen: false).add();  
      },  
    );  
  }  
}
```

- ▶ Recap
 - ▶ State inside widget – StatefulWidget and setState((){...})
- ▶ App State – use Provider package
 - ▶ ChangeNotifier – define ModelClass (extend or mixin ChangeNotifier)
 - ▶ ChangeNotifierProvider<ModelClass> - state holder in upper component
 - ▶ Consumer<ModelClass> - widget accesses state, gets recreated on state change
 - ▶ Provider.of<ModelClass>(context, listen: false) – access state in widgets, which are not UI dependent of state (action buttons for example)

- ▶ MultiProvider

```
return MultiProvider(  
  providers: [  
    ChangeNotifierProvider(builder: (_) => User()),  
    ChangeNotifierProvider(builder: (_) => Customer()),  
    ChangeNotifierProvider(builder: (_) => Group())  
  ],  
  child: MaterialApp(  

```

- ▶ ProxyProvider – generic provider that builds a value based on other providers

ProxyProvider<model1, model2, combinedModel>

```
ProxyProvider<Api, HomeModel>(  
  builder: (context, api, homeModel) => HomeModel(api: api),  
)
```


▶ THE END!